Journal of Physics: Conference Series **513** (2014) 032054 doi:10.1088/1742-6596/513/3/032054

Evaluation of Apache Hadoop for parallel data analysis with ROOT

S Lehrack, G Duckeck, J Ebke

Ludwigs-Maximilians-University Munich, Chair of elementary particle physics,
Am Coulombwall 1, D-85748 Garching, GER

E-mail: sebastian.lehrack@physik.uni-muenchen.de

Abstract. The Apache Hadoop software is a Java based framework for distributed processing of large data sets across clusters of computers, using the Hadoop file system (HDFS) for data storage and backup and MapReduce as a processing platform. Hadoop is primarily designed for processing large textual data sets which can be processed in arbitrary chunks, and must be adapted to the use case of processing binary data files which cannot be split automatically.
However, Hadoop offers attractive features in terms of fault tolerance, task supervision and control, multi-user functionality and job management. For this reason, we evaluated Apache Hadoop as an alternative approach to PROOF for ROOT data analysis. Two alternatives in distributing analysis data were discussed: either the data was stored in HDFS and processed with MapReduce, or the data was accessed via a standard Grid storage system (dCache Tier-2) and MapReduce was used only as execution back-end.
The focus in the measurements were on the one hand to safely store analysis data on HDFS with reasonable data rates and on the other hand to process data fast and reliably with MapReduce. In the evaluation of the HDFS, read/write data rates from local Hadoop cluster have been measured and compared to standard data rates from the local NFS installation. In the evaluation of MapReduce, realistic ROOT analyses have been used and event rates have been compared to PROOF.

1. Introduction

The handling of large data is a very common task in modern computing and many solution have been developed and examined so far. For processing large high energy physics experimental data, we investigated the Apache Hadoop[1]. It is structured as a two-component system: the Hadoop File System (HDFS) and MapReduce, a framework for processing data from the Hadoop file system. The HDFS is an open-source derivation of the Google file system (GFS); It aims for high redundance and availibility of data and cost reduce for hard drives.
The idea with the HDFS is to distribute the data redundant over many machines, called datanodes, in a network. This was achieved by splitting the data in blocks of a given size and then randomly storing these blocks mulitple times, each copy of a block on a different node. A central machine, the namenode, keeps track of every block and its location and ensures the replication of every block. In case of a disk failure or even the failure of a hole datanode, the random distribution of the blocks allows the namenode to restore the missing data blocks by recopying from another datanode. The difference to a normal backup system is that the data is still available during this restoring process.

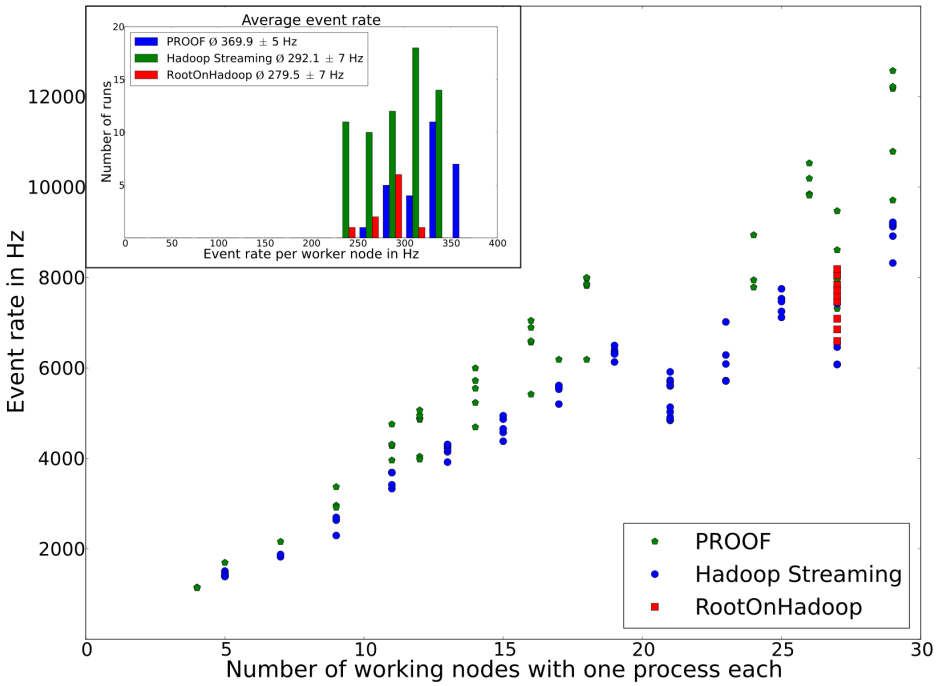

Figure 1. Calculated event rates for different cluster sizes and different analyses software. Inlay: Histogram of the average event rate per worker node.

Besides storing data, Hadoop offers a posibility to analyse and process data with MapReduce[1]. A map task will produce a lot of intermediate results, typically information of the same type and it can be done parallel. A reduce task will then merge these result e.g. by combining similar information or dropping unimportant parts.
In Hadoop, every machine in a cluster is both a datanode and a tasktracker. This means, that the data is stored and the analysis runs on the same physical machine, which reduces network traffic.
The MapReduce scheme is intended for data that can be splitted arbitrarily and processed separately. When using Hadoop for analysing experimental data i.e. binary ROOT data, this is not possible. Besides this, most ROOT files are larger than the typical block size of 64 MB of a HDFS cluster. So when using a Hadoop cluster for a ROOT analysis, these issues needed to be addressed.

2. Software

It made no sense to transfer already written ROOT analyses to JAVA, so we investigated Hadoops possibilities to run any executable or script. For this evaluation, we examined two, the Hadoop Streaming[2] and the RootOnHadoop Java classes from Stefano Alberto Russo[3].

2.1. Using RootOnHadoop

RootOnHadoop checks for every file consistency and location. It bypasses the Hadoop namenode and searches directly for the associated blocks on the local hard disks. If the data block is not available local, it downloads it via the Hadoop interface. Then the external map script is started

[1] Its name derived from the map and reduce function often used in functional programming.

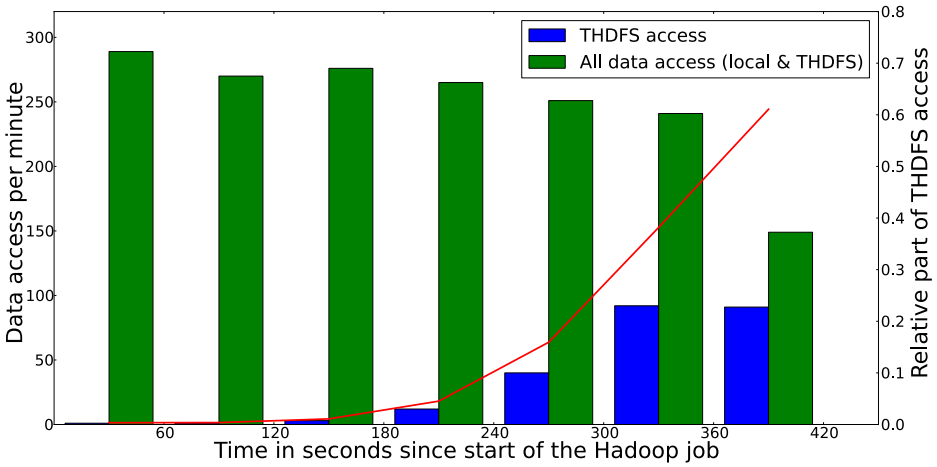

Figure 2. Time evolution of local and THDFS access during a RootOnHadoop run. In this Plot, 5 runs with 25 worker nodes and 350 data files have been accumulated.

which run the desired ROOT analysis. The output is logged and results are uploaded to the HDFS. A single reducer simply merges the results.
We optimized this procedure with the HDFS access from ROOT. With a patch[2], ROOT is able to read directly from the HDFS cluster and no data needs to be downloaded to the tasktracker.

2.2. Using Hadoop Streaming
The Hadoop Streaming is part of the Hadoop package and therefore available with every standard installation. It runs any script or executable, which will read and write the data and results from the standard input to the standard output. Mapper and reducer can be simply developed on the well known Linux terminal. We used the Hadoop Streaming in the following:

- A file list with file links to all data files is given as a input for the streaming and has therefore to be stored on the HDFS.
- The Hadoop Streaming splits this file list line-wise and sends each line to a separate map task.
- Each mapper starts the ROOT analysis directly with the file link.
- The results from the ROOT analysis are uploaded to the HDFS.
- As intermediate result, the file link to the resulting file on the HDFS is sent to the reducer.
- The reducer task merges the result files from the HDFS with the *hadd* command.

For this purpose, the experimental input data was not stored on HDFS but instead was read from a standard Grid storage system (dCache Tier2).

3. Measurements
We installed Hadoop on about 30 local workstations at the group for elementary particle physics at LMU Munich. The measurements of the time needed for a realistic ROOT analysis[3] were

[2] For the ROOT version we used (5.34/05), THDFSFile is not compatible with the Hadoop version (1.0.3). We applied a simple fix in the THDFSFile constructor to the argument list in the function call 'hdfsConnectAsUser'.
[3] A typical N-tuple analysis from LMU's ATLAS H $\rightarrow$ WW analysis group was applied. As input data, we used ATLAS 'SMWZ D3PD' files. The ROOT analysis performs a simple skimming of the data and fills the histograms for a cut-flow analysis.

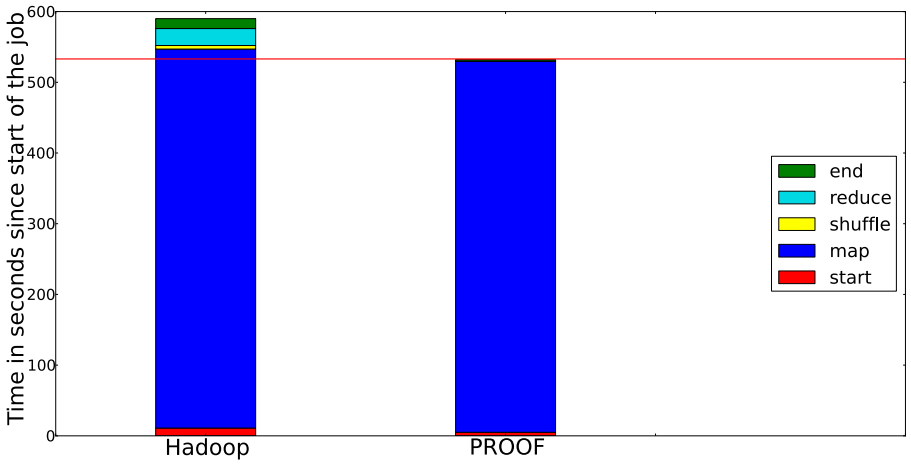

Figure 3. Comparison of the time needed for the different steps of a ROOT analysis for Hadoop Streaming and PROOF. The time was measured for a run on 19 worker nodes and 350 data files.

taken over night, analyzing the data with PROOF and with the Hadoop Streaming schema. Both were alternately and consecutively used. The required time was measured with the Linux *time* command. To evaluate the scaling behaviour, we repeated these measurements over a wide range of cluster sizes. In order to make the measured event rates comparable, we started the Hadoop cluster with the same worker nodes as PROOF. With the measured time, we calculated the event rate depending on of the number of worker nodes and the average event rate per worker node.
Event rates were also calculated for processing the same data set with the RootOnHadoop software. Since for that analysis the HDFS was used as the data storage for the ROOT input files, maintaining the same data set and calculating the event rates for different sizes of the cluster was not possible. Therefore, a full cluster with 27 nodes was used. Since the standard PROOF setup uses one process per working node, we changed the standard Hadoop setting accordingly for a better comparision.

4. Results

4.1. Comparison to PROOF

The calculated event rates as a function of cluster size for the different analyse software is shown in figure 1. The plot shows a linear scaling of the event rate with the number of working nodes for both, PROOF and the Hadoop Streaming schema, up to 30 working nodes. In our evaluation, we recognized no bounding behavior in the event rate. We also found out, that PROOF performed about 10% -20% better in terms of event rate compared to Hadoop. The RootOnHadoop schema achieved same event rates as the Hadoop Streaming, but still less than PROOF.
We also simulated a network fail during a running Hadoop job. In one test, we unplugged a running tasktracker from the network during a running task on this particular node. The clusters namenode and jobtracker recognized the missing datanode and tasktracker. Besides blacklisting and restoring the missing data blocks, the running Hadoop job was not harmed. By replugging the working node, the tasktracker and datanode was instantly reintegrated into the cluster. The results from that particular tasktracker were collected and processed further without taking notice of the temporal missing of the working node. In another test, we directly

shut down the tasktracker by terminating the corresponding process. The missing tasktracker was blacklisted by the jobtracker and no further task were provided to that particular node. Again, the Hadoop job continued without taking notice.
Hadoop is able to handle such situation due to the speculative execution. Every task is started as an attempt simultaneously on up to 3 different tasktracker. When it become obvious which tasktracker process the task the fastest, the other attempts are terminated. By that, slowly running task which will slow down the entire job are sorted out. Also error-prone tasktracker can be recognized and blacklisted.

4.2. Data locality

For an optimal usage of MapReduce, data should be processed from a local hard drive. Therefore we tested the data locality for the RootOnHadoop software. We also recognized that to the end of the Hadoop job, more and more task cannot access the file locally, because the job tracker scheduled the task without considering the data locality. In figure 2, the used data access methods, locally and via THDFS, during a Hadoop job is shown. For this histogram, the results of 5 Hadoop jobs have been accumulated and we used the RootOnHadoop software.

4.3. Time needed per analysis step

In figure 3, we plotted the measured time needed for the different analysis steps during a PROOF and a Hadoop job. The different steps were named after the corresponding steps in a Hadoop job. PROOF also reduces files, but likewise on each worker node before collecting the data for a final merge. Therefore, this step couldn't be resolved in this comparison. From this comparison, we see the rather large starting and ending overhead of the Hadoop job compared to PROOF. Furthermore, a final merge of the result files from all worker nodes is needed in the reduce phase of the Hadoop job, whereas PROOF can process multiple files in one ROOT process and merges them before this final merge.

5. Conclusion

In our evaluation, we tested and performed a ROOT analysis on binary, not-splittable data files and showed that we were able to use Hadoop as an alternative to PROOF. We tested the stability of Hadoop with simulated network failures and saw a good handling of missing intermediate results due to the speculative execution feature. Although the standard user management is simple, in a group like ours it eliminated prevalent administration effort.
In evaluating the performance of our cluster, we saw a 10% to 20% reduced event rate compared to PROOF. We concluded that this is due to the mentioned additional features and the fact, that PROOF nativly uses ROOT and is optimized for ROOT analyses, whereas Hadoop is primarily focused on processing large splitable data files, like text files. Besides, in our Hadoop schemas, for each data file, a new ROOT process has to be started whereas PROOF used one ROOT session to process multiple files. Therefore, starting overhead was minimized.
For an improvement on the data locality, it could be interesting to investigate the fair scheduler, which can lead to a nearly 100% data locality as reported in [3].

References

[1] Apache Hadoop URL `http://hadoop.apache.org`
[2] Apache Hadoop Streaming URL `http://hadoop.apache.org/docs/stable/streaming.html`
[3] Russo S A *"Running a typical ROOT HEP analysis on Hadoop/MapReduce"* (CHEP 2013, Beurs van Berlage, Amsterdam) 17. October 2013, conference presentation